This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Currently amended) A method for code from a first logical VM invoking a remote method of a second logical VM ~~executing a remote method~~, the method comprising when code from a first logical VM is invoking a remote method of a second logical VM:

~~determining whether each argument of a the remote method is a remote object;~~

wrapping each argument of the remote method when each argument is determined to be a remote object, wherein the first logical VM is associated with a first set of related code that are to be terminated together if misbehaving and the second logical VM is associated with a second set of related code that are to be terminated together if misbehaving, the first set of related code differing from the second set of related code;

copying each argument of the remote method when each argument is determined not a to be remote object;

invoking the remote method using each wrapped or copied argument;

~~determining whether a result of the invoked remote method is a remote object;~~

wrapping the result of the invoked remote method when the result is determined to be a remote object; ~~and~~

copying the result of the remote method when the result is determined not to be a remote object;

returning the wrapped or copied result only when an invocation thread associated with invoking the remote method of the second logical VM is not being terminated; and

throwing an exception on the remote method of the second logical VM when the invocation thread associated with invoking the remote method of the second logical VM is being terminated.

2. (Original) A method as recited in claim 1, wherein the argument or result is a remote object when its declared class implements a remote marker interface

3. (Currently amended) A method as recited in claim 1, wherein the result is only wrapped or copied when an invocation thread associated with invoking the remote method is not being terminated~~, the method further comprising throwing an exception on the remote method when the invocation thread is terminated~~.

4. (Original) A method as recited in claim 3, wherein wrapping each argument and result includes:

creating a wrapper object for the argument or the result and remembering an association between the wrapper object and the argument or the result when a wrapper object has not already been created; and

finding the wrapper object for the argument or the result based on a previous association between the argument or the result and the wrapper object when the wrapper object has already been created.

5. (Original) A method as recited in claim 4, wherein creating the wrapper object for each argument and the result includes:

finding or generating a remote stub class;

creating an instantiation of the remote stub class; and

setting a data member within the remote stub class to refer to the argument or the result.

6. (Original) A method as recited in claim 5, wherein generating the remote stub class includes:

generating a class name;

adding a method implementation for each method of a class of the argument or result being wrapped into a class definition array; and

remembering an association between the class and the remote stub class.

7. (Original) A method as recited in claim 6, wherein finding the remote stub class is based on a previously remembered association between a class and the remote stub class.

8. (currently amended) A method as recited in claim 1, wherein copying each argument and the result includes:

serializing each argument or the result into a byte array when the each argument or the result implements serialization;

deserializing the each argument or the result with respect to a target class loader associated with code that will use the copy of each argument or the result when the each argument or the result implements serialization;

implementing a failure process when the each argument or the result does not implement serialization.

9. (Currently amended) A computer readable medium containing computer codes for executing a remote method, the computer readable medium comprising:

computer code for when code from a first logical VM is invoking a remote method of a second logical VM, wrapping each argument of the remote method when each argument is determined to be a remote object, wherein the first logical VM is associated with a first set of related code that are to be terminated together if misbehaving and the second logical VM is associated with a second set of related code that are to be terminated together if misbehaving, the first set of related code differing from the second set of related code;

computer code for wrapping each argument of the remote method when each argument is determined to be a remote object;

computer code for copying each argument of the remote method when each argument is determined not a to be remote object;

~~computer code for determining whether a result of the invoked remote method is a remote object;~~

computer code for invoking the remote method using the wrapped or copied argument(s);

computer code for wrapping the result of the invoked remote method when the result is determined to be a remote object; ~~and~~

computer code for copying the result of the remote method when the result is determined not to be a remote object; and

computer code for returning the wrapped or copied result only when an invocation thread associated with invoking the remote method of the second logical VM is not being terminated; and

computer code for throwing an exception on the remote method of the second logical VM when the invocation thread associated with invoking the remote method of the second logical VM is being terminated..

10. (Original) A computer readable medium as recited in claim 9, wherein the argument or the result is a remote object when its declared class implements a remote marker interface.

11. (Currently amended) A computer readable medium as recited in claim 9, wherein the result is only wrapped or copied when an invocation thread associated with invoking the remote method is not being terminated~~, the computer readable medium further comprising computer code for throwing an exception on the remote method when the invocation thread is terminated~~.

12. (Previously presented) A computer readable medium as recited in claim 11, wherein wrapping the argument or the result includes:

creating a wrapper object for the argument or the result and remembering an association between the wrapper object and the argument or the result when a wrapper object has not already been created; and

finding the wrapper object for the argument or the result based on a previous association between the argument or the result and the wrapper object when the wrapper object has already been created.

13. (Original) A computer readable medium as recited in claim 12, wherein creating the wrapper object for the argument or the result includes:

finding or generating a remote stub class;
creating an instantiation of the remote stub class; and
setting a data member within the remote stub class to refer to the argument or the result.

14. (Original) A computer readable medium as recited in claim 13, wherein generating the remote stub class includes:

generating a class name;
adding a method implementation for each method of a class of the argument or result being wrapped into a class definition array; and
remember an association between the class and the remote stub class.

15. (Original) A computer readable medium as recited in claim 14, wherein finding the remote stub class is based on a previously remembered association between a class and the remote stub class.

16. (Currently amended) A computer readable medium as recited in claim 9, wherein copying each of the argument or the result includes:
serializing each argument or the result into a byte array when the each argument or the result implements serialization;
deserializing each argument or the result with respect to a target class loader associated with code that is using the copy of each argument or the result when the each argument or the result implements serialization;
implementing a failure process when the each argument or the result does not implement serialization.

17. (Currently amended) A computer system for executing a remote method, the computer system comprising:

a memory; and

a processor coupled to the memory,

wherein at least one of the memory and the processor are adapted for when code from a first logical VM is invoking a remote method of a second logical VM:

~~determining whether each argument of a remote method is a remote object;~~

wrapping each argument of the remote method when each argument is determined to be a remote object, wherein the first logical VM is associated with a first set of related code that are to be terminated together if misbehaving and the second logical VM is associated with a second set of related code that are to be terminated together if misbehaving, the first set of related code differing from the second set of related code;

copying each argument of the remote method when each argument is determined not a to be remote object;

invoking the remote method using each wrapped or copied argument;

~~determining whether a result of the invoked remote method is a remote object;~~

wrapping the result of the invoked remote method when the result is determined to be a remote object; ~~and~~

copying the result of the remote method when the result is determined not to be a remote object;

returning the wrapped or copied result only when an invocation thread associated with invoking the remote method of the second logical VM is not being terminated; and

throwing an exception on the remote method of the second logical VM when the invocation thread associated with invoking the remote method of the second logical VM is being terminated.

18. (Previously presented) A computer system as recited in claim 17, wherein the argument or the result is a remote object when its declared class implements a remote marker interface.

19. (Currently amended) A computer system as recited in claim 17, wherein the result is only wrapped or copied when an invocation thread associated with invoking the remote method is not being terminated~~, the computer readable medium further comprising computer code for throwing an exception on the remote method when the invocation thread is terminated~~.

20. (Previously presented) A computer system as recited in claim 19, wherein wrapping the argument or the result includes:

creating a wrapper object for the argument or the result and remembering an association between the wrapper object and the argument or the result when a wrapper object has not already been created; and

finding the wrapper object for the argument or the result based on a previous association between the argument or the result and the wrapper object when the wrapper object has already been created.

21. (Previously presented) A computer system as recited in claim 20, wherein creating the wrapper object for the argument or the result includes:

finding or generating a remote stub class;
creating an instantiation of the remote stub class; and
setting a data member within the remote stub class to refer to the argument or the result.

22. (Previously presented) A computer system as recited in claim 21, wherein generating the remote stub class includes:

generating a class name;
adding a method implementation for each method of a class of the argument or result being wrapped into a class definition array; and
remember an association between the class and the remote stub class.

23. (Previously presented) A computer system as recited in claim 22, wherein finding the remote stub class is based on a previously remembered association between a class and the remote stub class.

24. (Previously presented) A computer system as recited in claim 17, wherein copying each of the argument or the result includes:

serializing each argument or the result into a byte array when the each argument or the result implements serialization;

deserializing each argument or the result with respect to a target class loader associated with code that is using the copy of each argument or the result when the each argument or the result implements serialization;

implementing a failure process when the each argument or the result does not implement serialization

25. (New) A method as recited in claim 1, wherein the first set of related code of the first logical VM is assigned to a first protection domain and the second set of related code of the second logical VM is assigned to a second protection domain that differs from the first protection domain.

26. (New) A computer readable medium as recited in claim 9, wherein the first set of related code of the first logical VM is assigned to a first protection domain and the second set of related code of the second logical VM is assigned to a second protection domain that differs from the first protection domain.

27. (New) A computer system as recited in claim 17, wherein the first set of related code of the first logical VM is assigned to a first protection domain and the second set of related code of the second logical VM is assigned to a second protection domain that differs from the first protection domain.